syslog: The UNIX System Logger

John Fenwick
Development Engineer
Enterprise Systems Division
Hewlett-Packard Company
Cupertino, California USA
fenwick@cup.hp.com
Tel. 408-447-4976

1. Introduction

syslog is a powerful and easily configurable UNIX system resource. Available since the earliest releases of BSD UNIX, it is now supported on most UNIX versions. Designed to be the UNIX system logging facility, syslog has always offered not just local logging to files but also remote logging over the network. Network communication via standard protocols permits syslog to operate across platforms.

This paper discusses the elements making up syslog - the daemons, libraries, commands, data structures, and communication paths. Each communication path available to syslog is reviewed. We describe the appropriate use of sockets, pipes, and files. syslog message syntax and format is covered. Lastly, the rules to configure and operate the syslogd daemon are reviewed.

Several examples illustrating syslog usage are reviewed. HP-UX kernel message recovery through syslog using the /dev/klog interface is described. Program interfaces for C-code and shell scripts are illustrated. Examples using remote logging and cross-platform operation are shown.

We close with a discussion of syslog limitations, and a comparison of syslog to related UNIX facilities.

1.1 System Logging in UNIX

Logging and error reporting within UNIX has been unnecessarily complex. The following examples illustrate typical approaches for logging messages and reporting errors to the system console.

```
in a shell script:
% ./mydaemon 2> /dev/console &    # stderr sent to console

in C-code:
fp_console = fopen(“/dev/console”,”w”)
...
if ( ! stat(”filename”,&stat_buf) )
    printf(“file stat call successful”);
else                                /* error output to console */
    fprintf(fp_console,”Error: cannot open file”);
```

While this may be acceptable for output written directly to your process window, these approaches have potential drawbacks for console I/O:

- the message will be lost on a console-less system

- the message may scroll off the console and be lost
- the operator may not be at console for immediate action
- a user logged in at the console may receive a message that is confusing or inappropriate
- the message will not be logged for future review
- console output may become garbled by competing processes
- console screen appearance and formatting may be overwritten

In general, system log and error messages should not be written directly to the system console, except under the most urgent of circumstances. For example, a kernel crash and the display of system state information may still be directed to the system console.

To remedy non-uniform message reporting and address problems noted above, early releases of the BSD-UNIX operating system introduced the syslog message reporting system. syslog was then ported to major UNIX system implementations as features of BSD-UNIX were adopted. syslog has been implemented in HP-UX since the early releases of the operating system. Recent work has been done to syslog in the 10.X releases for performance and standards issues.

2. Architecture of syslog

The syslog system consists of the following components:

- a message format specification <syslog.h>

  syslog messages are encoded as ASCII strings. Message strings are created throughout the UNIX system. Messages are created at one of a set of possible levels; by setting a threshold one can direct all messages at this or a higher level to given locations. Definitions for syslog messages are in the include file <syslog.h>.

- a set of calls for creating those messages syslog(3c), logger(1)

  Most users create syslog messages though one of the standard interfaces to syslog. The library call syslog(3c) (contained within libc) is a C-code interface to creating message strings. syslog(3c) behaves somewhat like the standard printf() interface. From the command line or within a shell script one can invoke the command logger(1) to create syslog messages.

- a set of locations from which messages can be read /dev/log, /dev/klog, UDP port 514

  Messages come into syslog from various paths. Every syslog message must be directed to one of a number of communication paths that are read by the syslogd daemon. These communication channels can include:

a UNIX pipe	/dev/log	
a special kernel interface	/dev/klog	(character special file)
an Internet domain socket	UDP port 514	

- a daemon that reads from these locations syslogd

  The syslogd daemon is a user-space process. syslogd waits for incoming messages and directs the message to possible output locations. A message that is logged (to a file, a logged-on user, or the console) is also formatted and tagged with the system name. The syslogd daemon may also do some filtering of messages; this might be done to prevent repeated messages from flooding the system.

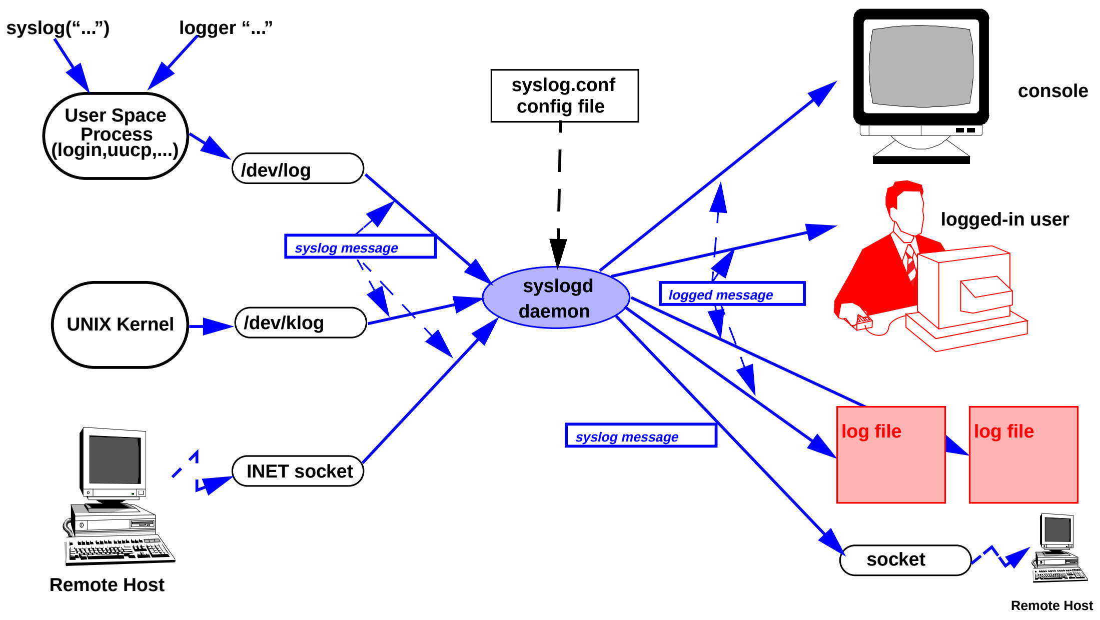
- a set of locations to which messages can be directed files, users, etc.

  This set of locations may include sockets, files, the system console, and logged-on users. Messages may be directed to one or more of these locations.

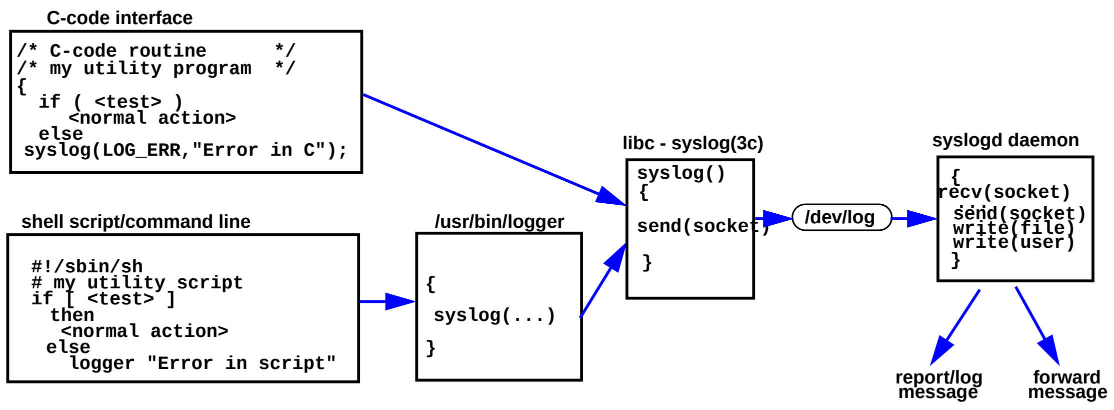
- rules for how messages are directed. /etc/syslog.conf

  This set of configuration rules determines how messages are logged and the locations to which they are logged. The rules are set by the System Administrator in the file /etc/syslog.conf.

The flow of messages through the parts of the syslog system is pictured in the following diagram.



Software developers who are creating their own syslog messages will be interested in the path for user-created syslog messages. User messages are typically created through either the syslog(3c) C-code interface or the logger(1) command. These messages are written to the pipe /dev/log, from which they are then read by the syslogd daemon, which directs them to configured output locations. This flow of messages, and sample code fragments for creating these messages, are shown in the following diagram.



```
/* C-code routine      */
/* my utility program  */
{
  if ( <test> )
     <normal action>
  else
 syslog(LOG_ERR,"Error in C");
```

```
  #!/sbin/sh
  # my utility script
  if [ <test> ]
    then
      <normal action>
    else
      logger "Error in script"
```

```
{
 syslog(...)
}
```

```
syslog()
{
 ...
send(socket)
 }
```

```
 {
recv(socket)
 ...
 send(socket)
 write(file)
 write(user)
 }
```

3. Message format

One of the unique characteristics of syslog is that every message is created and logged in the form of a plaintext ASCII string. Messages are then directed through pipes and sockets, written to log (ordinary) files, and displayed to users using only standard string handling operations. This presents an easy to use and well understood interface, but has a few performance implications that will be discussed later.

Through this discussion we will maintain a distinction between what we will refer to as a *syslog message* and a *logged message*. A syslog message is the actual ASCII string that is sent to or read from the pipes and sockets that make up the syslog system. It is useful to review this message format to understand some of the capabilities and limitations of syslog.The logged message is the plaintext ASCII string that is actually written to a log file, or to a user at her terminal. This is the string the system administrator or an ordinary user will actually see.

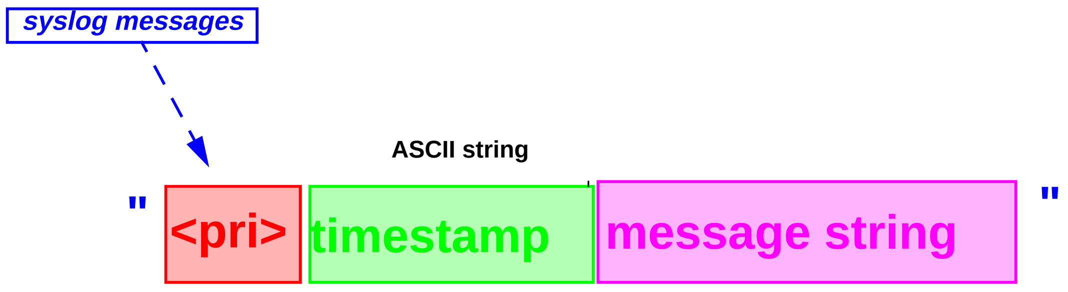
syslog message format is defined in the include file <syslog.h>. In this file are the definitions for message priority (level and facility) and the flags used to configure the syslog interfaces. Function prototypes for the syslog library calls were added to <syslog.h> for the HP-UX 10.X releases.

3.1 Components of syslog messages

A syslog message is an ASCII string that consists of

- a message priority
- a timestamp
- the message string

A diagram of a syslog message is shown below.



priority: an ASCII integer that is a combination of:
level ordered levels 0-7
facility the subsystem originating the message

timestamp: an ASCII string for time of day
Format: Mth DD HH:MM:SS
Note: this is the time of day on the *originating* system

message string: text of the message to log

Priority is encoded as an ASCII string enclosed by the angle brackets < and > at the beginning of the string. Message priority is the ASCII integer encoding of an 8-bit quantity. This quantity is a combination of a 3-bit field (bits 0 through 2) used for message level and a 5-bit field (bits 3 through 7) used for the message facility. Thus message priority level can have 8 possible values, and message facility up to 32 possible values.

Message levels are defined as an ordered list. If one has set a threshold at a given level one will receive all messages at this or any higher level. Thus, if you have configured your syslog to log all messages tagged LOG_WARNING, you will also log all messages of higher levels, such as LOG_ERR, LOG_CRIT, and so on. The defined message levels, from highest level to lowest, are as follows:

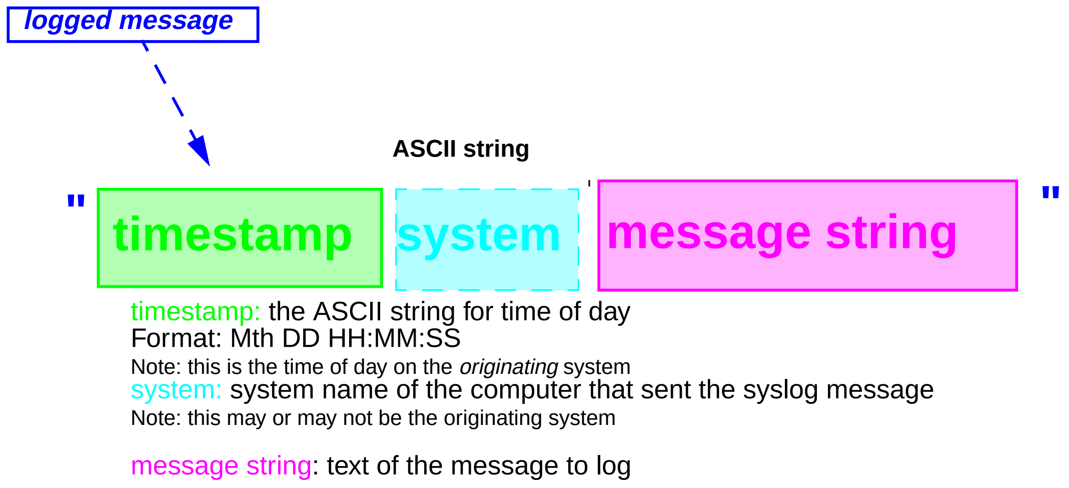
LOG_EMERG	0	Kernel panic
LOG ALERT	1	Condition needing immediate attention
LOG_CRIT	2	Critical conditions
LOG_ERR	3	Errors
LOG_WARNING	4	Warning messages
LOG_NOTICE	5	Not an error, but may need attention
LOG_INFO	6	Informational messages
LOG_DEBUG	7	When debugging a system

The message facility is a tag to identify the originating subsystem of the message. Facility tags are defined in <syslog.h>, some are reserved for the OS, and others are available for users and application developers. The following message facility tags are defined:

LOG_KERN	(0<<3)	kernel messages
LOG_USER	(1<<3)	random user-level messages
LOG_MAIL	(2<<3)	mail system
LOG_DAEMON	(3<<3)	system daemons
LOG_AUTH	(4<<3)	security/authorization messages
LOG_SYSLOG	(5<<3)	messages generated internally by syslogd
LOG_LPR	(6<<3)	line printer subsystem
LOG_NEWS	(7<<3)	messages generated by the news system
LOG_UUCP	(8<<3)	messages generated by the UUCP system
LOG_CRON	(9<<3)	messages generated by the cron daemon
other codes through 15 are reserved for system use		
LOG_LOCAL0	(16<<3)	reserved for local use
LOG_LOCAL1	(17<<3)	reserved for local use
LOG_LOCAL2	(18<<3)	reserved for local use
LOG_LOCAL3	(19<<3)	reserved for local use
LOG_LOCAL4	(20<<3)	reserved for local use
LOG_LOCAL5	(21<<3)	reserved for local use
LOG_LOCAL6	(22<<3)	reserved for local use
LOG_LOCAL7	(23<<3)	reserved for local use

3.2 Logged Messages

The logged message is the ASCII string that will be written by the syslogd daemon to a user, to the console, or to a log file. The logged message is also a plaintext ASCII string, but of somewhat different format. A diagram of the logged message is given in the following figure.


logged message
ASCII string
"
timestamp
system
message string
"
timestamp: the ASCII string for time of day
Format: Mth DD HH:MM:SS
Note: this is the time of day on the originating system
system: system name of the computer that sent the syslog message
Note: this may or may not be the originating system
message string: text of the message to log

The message priority field is removed; the information in this field was used by syslogd to direct the syslog message according to the configuration rules set in /etc/syslog.conf. The timestamp is the same ASCII-encoded date/time string as before. A new “system” field is next in the string: this is the system name (equivalent to “uname -n”) that has sent the message. This is the local system if the message was locally generated, or may be the name of a remote system communicating over an Internet socket. The message string is the actual text of the message; useful fields such as process id and a message prefix may be inserted in this message by using the openlog() function.

4. Using syslog

4.1 Creating syslog messages syslog(3c), logger(1)

Most users and software developers will create syslog messages through the standard interfaces syslog(3c) and logger(1). The syslog library functions openlog(), closelog(), and setlogmask() are contained in the libc library; function prototypes are defined in <syslog.h>. The logger command (/usr/bin/logger) provides similar functionality from the command line or in a shell script.

The syslog call provides an interface with printf()-like string handling to syslog. The syslog call is used as follows:

```
syslog(priority,”message_string”)
```

Message priority is a combination of a syslog level and facility tag. The message string has a printf-like interface: one can log ASCII strings using standard %-semantics. In addition, the %m entry will print the global errno for the originating process. syslog() will return an error if /dev/log cannot be opened. For the 10.X releases syslog() was modified to retry the write to /dev/log if it is busy, with a 1/10 second delay for up to 20 attempts.

The function openlog() provides a useful method to control the logging process and set default fields in the message strings. The function is called as follows:

```
openlog("<identifying_string>",log_options,default_facility)
```

where the following parameters can be set:

"identifying_string" is a string that is prepended to each syslog message

log_options can be one or more of the following:

LOG_PID	log process ID with each message (very useful)
LOG_CONS	write to console if unable to send to syslogd
LOG_NDELAY	open connection to syslogd immediately
LOG_NOWAIT	no wait for children logging messages to console

default_facility assigns a facility tag to identify the originating subsystem (very useful)

The following calls are also part of the syslog library interface:

```
setlogmask(maskpri)
```

sets the process log priority mask to maskpri and returns the previous priority mask value

```
closelog()
```

closes the log file

From the command line or a shell script the command logger(1) can be used. logger parses the command line arguments and calls syslog(3c).

```
logger [-t tag] [-p priority] [-i] [-f file | message ]
```

-t tag	mark entry with this tag (default is getlogin())
-p priority	facility.level pair
-i	log the process ID with each message
-f	log from the contents of the file
message	message to log
	(log from stdin (message) or file (-f))

4.2 Example code

An example of configuring and using syslog is given in the following diagram. A developer first configures default settings for message strings using openlog(), and then logs messages to syslog() as needed.

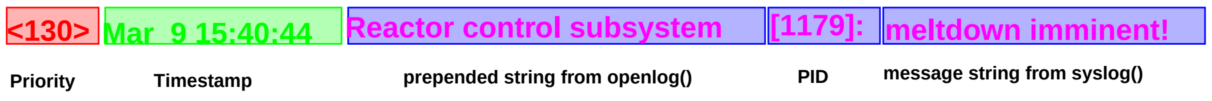
Configure the log file

```
openlog("Reactor control subsystem",LOG_PID|LOG_CONS,LOG_LOCAL0);
```

Send a message:

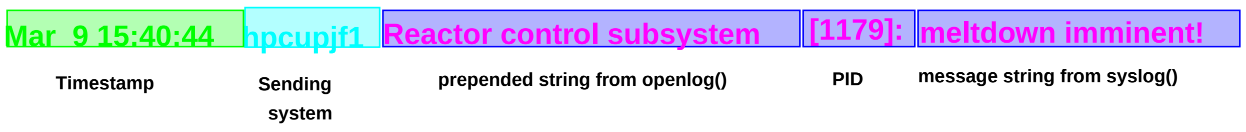
```
syslog(LOG_ALERT,"meltdown imminent!");
```

Format of the syslog message:



```
<130>Mar  9 15:40:44 Reactor control subsystem [1179]: meltdown imminent!
```

Format of the logged message:


Mar 9 15:40:44
hpcupjf1
Reactor control subsystem
[1179]:
meltdown imminent!
Timestamp
Sending system
prepended string from openlog()
PID
message string from syslog()

4.3 Configuring syslog

When the operating system is booted, syslog is brought up as a user space process. syslog is started early in the start sequence so that subsequent services can use syslog logging if needed. When the syslogd daemon is started, it reads the configuration file /etc/syslog.conf for its initial configuration. One can reconfigure syslogd and cause it to re-read its configuration file by sending it the SIGHUP signal (“**`kill -1 <syslogd_pid>`**”). The rules in /etc/syslog.conf are set by the system administrator.

Each non-comment line of syslog.conf is read as a configuration directive. The format of the directives is given in the following diagram.

selector
<tab>
action

Selector: List of priority (Facility.Level) specifiers, semicolon separated
Facility is the subsystem originating the message
wildcard ***** selects all facilities
Level sets a threshold for which messages are logged
HP-UX: must select a level, no wildcard allowedl
Examples:

```
user.error <tab> /var/adm/syslog/syslog.log  # log non-critical errors
*.emerg    <tab> /dev/console               # emergencies to console!
```

Priority tags can be one of the following ASCII strings: ”emerg” | ”panic”, “alert”, “crit”, “error” | ”err”, “warning” | ”warn”, “notice”, “info”, “debug” ; these correspond to the priority tags used in syslog(3c). Similarly, facility tags can be one of the following strings: ”kern”, “user”, “mail”, “daemon”, “auth” | ”security”, “mark”, “syslog”, “lp” | ”lpr”, “local[0-7]”.

Examples of possible selectors include:

```
user.error <tab> /var/adm/syslog/syslog.log  # non-critical
*.emerg  <tab> /dev/console      # emergencies to console
```

The action field specifies where the message is to be directed. Possible locations include:

file name	can be regular file, for example a log file
device file	for example. the console at /dev/console
list of users	write to a user if logged in
*	write to all users currently logged in
@remote-host	forward to syslogd on remote system

Here is an examples of selectors one might use:

```
*.emerg <tab>  /dev/console           #might be crashing!
*.emerg <tab>  *                      # let users know too
```

```
*.alert <tab>  root,fenwick           #let root know ALERTS
*.info  <tab>  /var/adm/syslog/syslog.log#standard log file
```

In HP-UX 10.X, the standard location for logging syslog messages is the file /var/adm/syslog/syslog.log. The log file from the prior bootup or invocation of syslog is preserved as the file /var/adm/syslog/OLDsyslog.log.

4.3.1 Logging to Remote Hosts

One of the most useful features of syslog is that syslogd can write to an Internet-domain socket connected to a socket on a remote host. The local syslogd daemon opens a socket and writes messages to it. A remote syslogd daemon receives the message. This socket is bound to Port 514/udp, which is reserved for syslog in /etc/services. Multiple hosts can be specified, and remote hosts can be chained together. If one is chaining remote hosts together, one should be aware that the logged message will list the system name of the *sending* (which may or may not be the *originating*) system.

Here is an example of how one would configure syslog for remote logging to the system “admin” in the file /etc/syslog.conf:

```
# sample /etc/syslog.conf file
*.error <tab> @admin.cup.hp.com  # forward to Admin. station
```

4.4 System-generated syslog messages

Many UNIX subsystems have been converted to log their messages through syslog. One can now review most kernel-generated messages through the syslog logfile. Kernel-generated messages are read through the character-special device file /dev/klog. Most messages generated by the kernel in bootup and operation are logged to syslog. In fact, most of the information presented on the kernel bootup screens is logged. One can find the same information in the syslog log file that one can find in the kernel message buffer - there is no longer any reason to use the dmesg(1) command!

Many of the Internet services (such as the inetd server and sendmail) have also been configured to log errors to syslog. The nettl logging service used in network drivers and services is a different logging routine, but it announces its own startup through syslog.

Since syslog operates over TCP/IP networks, networked peripherals can also use syslog services. For example, HP printers that use the JetDirect network interface card can be configured to report error conditions to syslog. This may be a convenient way to detect and report common printer errors.

5. Considerations when using syslog

Now that we have covered the structure and use of syslog, it is interesting to consider some of the limitations of the syslog system. Many of the features of syslog (small, easy to understand, uses plaintext strings) also result in some definite performance considerations.

5.1 Internationalization

The syslog system is not internationalized, and has no explicit provisions for internationalization. There are only a few status or error messages that are generated within the syslog libraries or daemon; each message is displayed in English.

Furthermore, since the messages that syslog forwards on behalf of its callers are handled entirely as plaintext strings, there is no direct way a message could be generated in one language and logged in another language. To syslog, the message is just an array of characters. One could get around this limitation by first translating an error message and then calling syslog, but after translation the message is just another string of bytes. Although the forwarding of messages through sockets could easily cross a country boundary, the message string would still be as composed by the generator.

5.2 Performance

Since the syslogd daemon is a user space process and communicates with other processes via IPC, message throughput takes appreciable time and is not suited for rapid logging. It will typically take some fraction of a second to generate and log a message on the local system. There is overhead due to the use of ASCII strings throughout syslog. For this reason syslog is not suited for logging events that occur many times per second. Kernel services often use their own logging facilities that use special instrumentation points within the kernel drivers. For example, the nettl logging facility allows one to log network events down to the packet level.

The syslog time stamp is recorded to a one-second precision. One may receive multiple messages with the same time stamp; one is guaranteed that the order in which they are logged is the same as the order in which they are received. In a UNIX system it is possible to fill a pipe or socket by writing to it more rapidly than it can be read or drained. When this happens the write() operation will return -1 and one should retry the write. For the 10.X release the syslog(3c) library call was modified to automatically attempt to retry a write to busy pipe; syslog will retry the write for 20 times with a 100 msec. timeout between writes. If the write still fails, syslog() will return -1.

Finally, the UDP protocol chosen for syslog communication does not provide reliable delivery of messages. On a busy network UDP packets may be dropped, and a syslog message may then be lost. One instance when this can be seen is when one directs syslog traffic from a high speed subnet across a slower bridge.

5.3 Functionality changes in recent releases

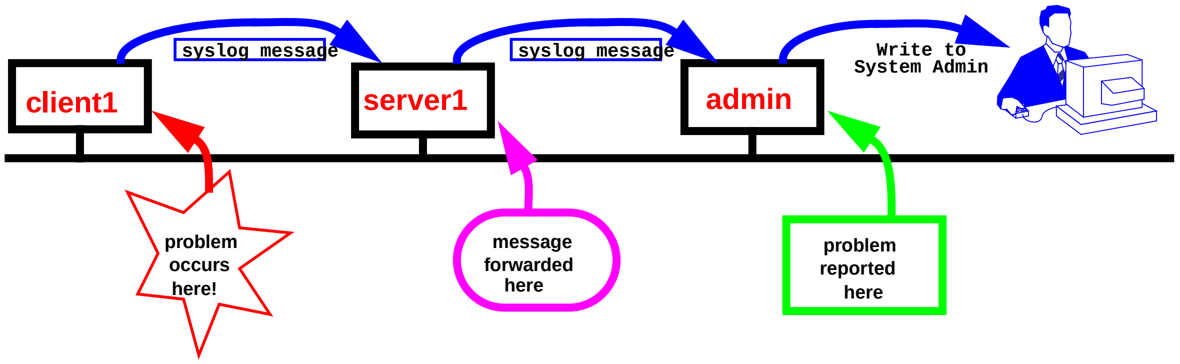
Logging of kernel messages through the /dev/klog interface was added to HP-UX in the 9.04 release. Every kernel message that one could retrieve from the kernel message buffer (accessible via the dmesg(1m) command) is logged in the file /var/adm/syslog/syslog.log. This is a more reliable way to retrieve these messages, since they will be reliably logged and not overwritten, as is the case in the finite-sized kernel message buffer.

For the 10.X releases the syslog library interfaces were made thread-safe.

5.4 Chaining remote syslog messages

The ability of syslog to easily direct messages across TCP/IP networks to a remote syslogd is a powerful capability. However, since syslog logs only limited information in the message string, forwarded messages can become confusing.

Here is an example of how this might occur. Consider a system where one has directed syslog messages across two remote hosts (or upwards through a two-level hierarchy). A syslog message is generated on a low-level system “client1”, is forwarded to a mid-level system “server1”, and is finally directed to the System Administrator at the central system “admin”. The System Administrator will receive a message that will state “server1: <error message>”, whereas the error actually occurred on system “client1”. One could get around this potentially confusing system be including an explicit system name (as returned from “uname -n”) in the message that is originally generated.



5.5 Security considerations

Since syslogd runs as a root daemon, special care must be taken in the daemon to prevent an unscrupulous user from breaking into the daemon and getting unauthorized root access. Early versions of syslog were susceptible to the “stack bombing” attack. In this attack one gets the syslogd daemon to log an overly long message that, when copied into the syslogd process stack, causes stack overrun and can result in unauthorized root access. The underlying weakness exploited in this sort of attack (and other “stack bombing” attacks against other root process daemons) is faulty code that does not do bounded writes when copying arguments onto the stack.

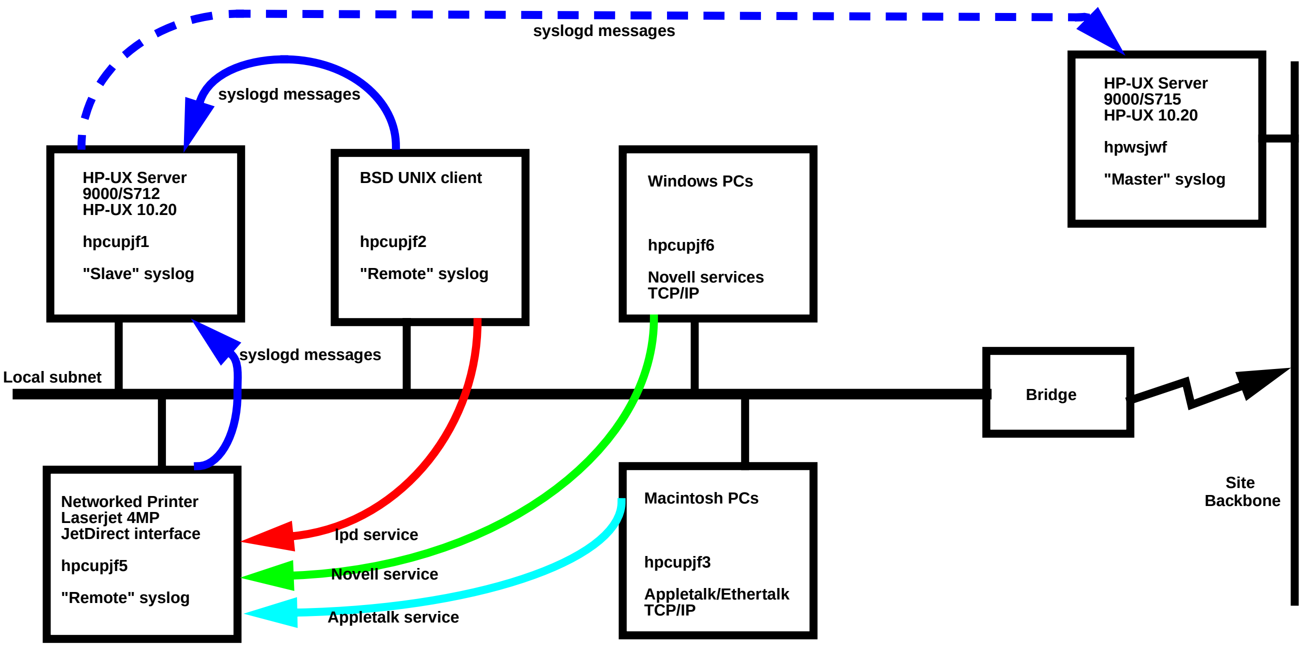
Details of this type of attack are described in the CERT Advisory CERT CA-95:13 and in HP Security Bulletin HPSBUX96092-029, dated Feb. 7, 1996. HP-UX has issued a series of patches to close this security loophole, and these are listed below. Please make sure that the appropriate patches are installed on your HP-UX installations.

PHCO_6595 (series 700/800, HP-UX 10.0, 10.01)
PHCO_6598 (series 800, HP-UX 9.X)
PHCO_6597 (series 700, HP-UX 9.0X)
PHCO_6224 (series 300/400, HP-UX 9.X)
PHCO_6595 (series 700/800, HP-UX 10.0, 10.01)
PHCO_6598 (series 800, HP-UX 9.X)
PHCO_6597 (series 700, HP-UX 9.0X)
PHCO_6224 (series 300/400, HP-UX 9.X)

6. Example: Cross-platform Operation

Here is an illustrative example of how syslog can be easily used to monitor system resources in a distributed, heterogeneous environment. The system consists of several types of computers (HP-UX and other variants of UNIX, PCs running Windows 95 and Windows NT, and Macintosh PCs. System peripherals, in this case the shared printers, are distributed on the network. The local subnet, and communication to the JetDirect printers in particular, is supporting several type of protocols including

TCP/IP, Novell services, and Appletalk. The local subnet is connected via a bridge (potentially an ISDN or SLIP connection) to the central administrative system.



Since the JetDirect printers support syslog reporting, one can easily configure the printers to report all error conditions to the central reporting system. The System Administrator will be notified when each printer does something as obvious as running out of paper, and will probably know and diagnose the error before the local users have become aware of it!

7. Conclusions

We have reviewed the structure of the syslog system and seen how the simple architecture of syslog offers both advantages and limitations. Some features of syslog to be aware of include the following.

syslog is a standard system logging tool across UNIX systems. Every major UNIX implementation supports syslog, and all syslogs talk to one another without difficulty.

Because of this easy interoperability, syslog works well in a distributed or client-server environment. One can easily forward and collect syslog messages to central reporting nodes.

syslog is easy to configure and does not consume a lot of system resources. The code for both the syslogd daemon and syslog() library interfaces is small, and the daemon consumes small resources when it is running.

Since the timestamp precision of syslog is only to one second, and since time is required for process swaps and inter-process communication, syslog is acceptable for user process logging and some forms of kernel logging. It is not suited for the rapid logging one may need when instrumenting kernel drivers.

Finally, syslog has some limitations that one should remain aware of. System name logging is potentially confusing. The logging service is not internationalized, and has no provisions for handling internationalized messages.